Multi-core, NUMA, Heterogeneous...Oh my!

Jeff Keasler

Aug 9, 2007

This work was performed under the auspices of the U.S. Dept. of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

UCRL-PRES-233978

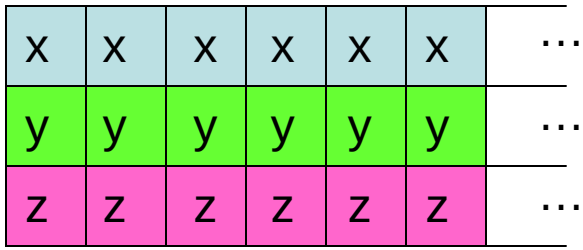
Outline

- Data Structure Choices
- Performance of Data Structure Choices
- Vector Extensions
- Advanced Features of Vector Extensions
- Conclusion

Fundamental Data Layouts

- Array-Like
  - double x[10000] ;
    double y[10000] ;
    double z[10000] ;

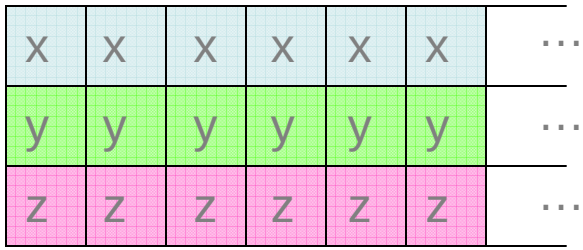
Memory Interleave

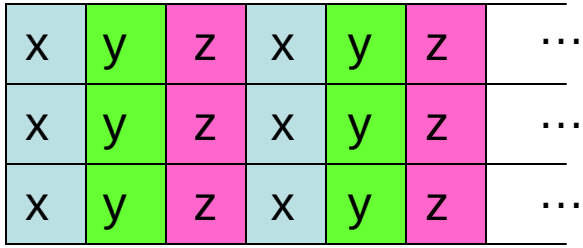


Fundamental Data Layouts

- Array-Like
  - double x[10000] ;
    double y[10000] ;
    double z[10000] ;
- Struct-Like
  - struct coord {
    double x, y, z ;
    } mesh[10000] ;

Memory Interleave

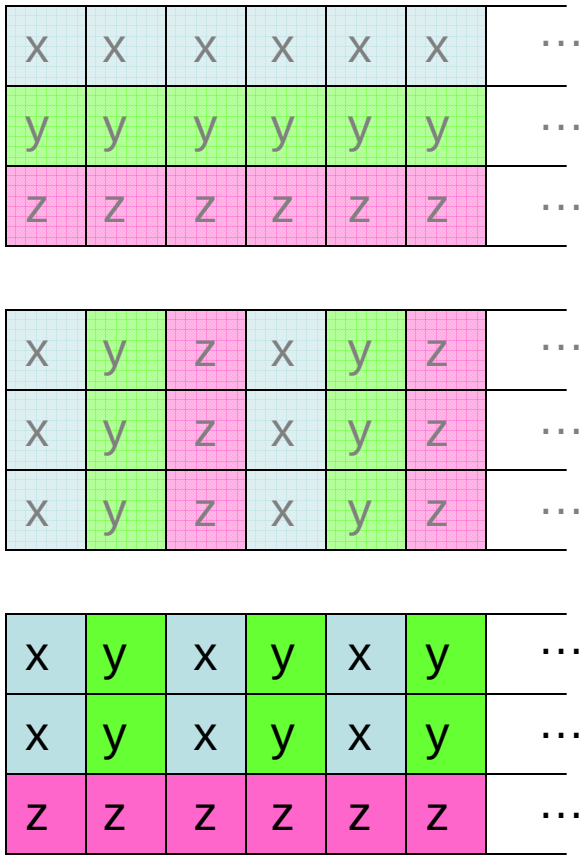




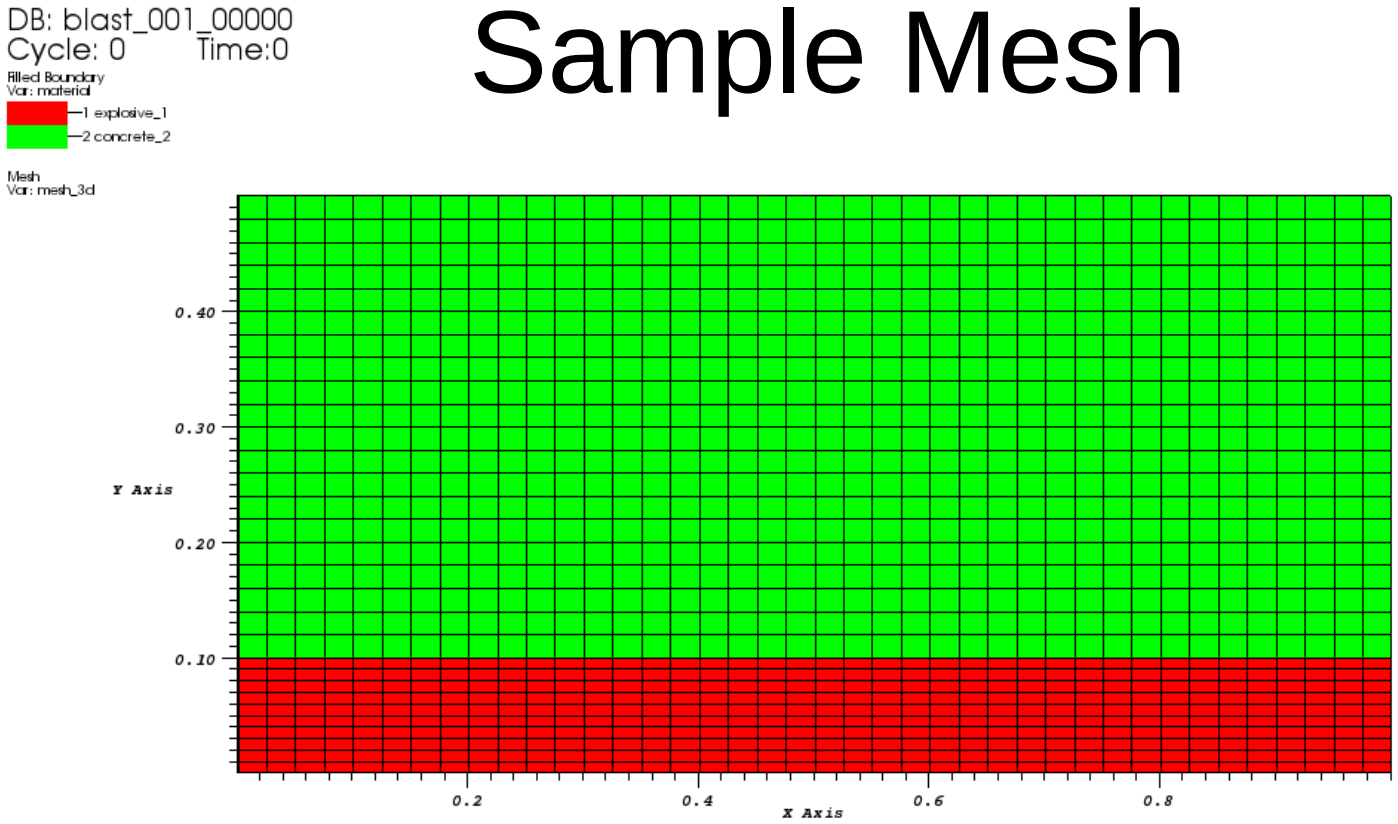
Fundamental Data Layouts

- Array-Like
  - double x[10000] ;
    double y[10000] ;
    double z[10000] ;
- Struct-Like
  - struct coord {
    double x, y, z ;
    } mesh[10000] ;
- Clustered-Struct
  - struct coord {
    double x,y ;
    } coord[10000] ;
    double z[10000] ;

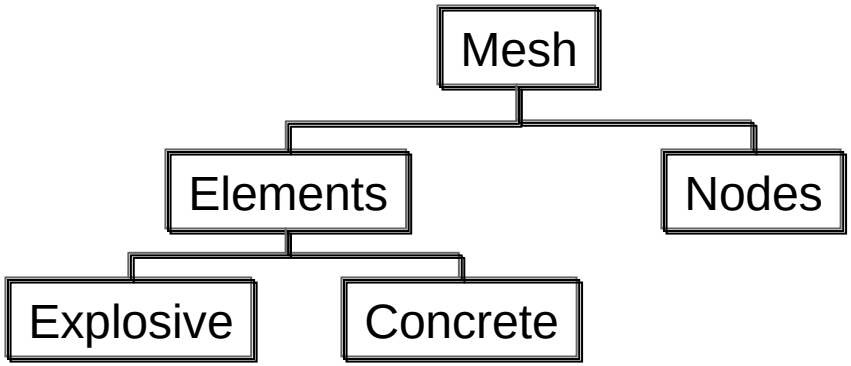
Memory Interleave



Sample Mesh



Topological Structure:



Array-like access

```
real8 quarterDelta = 0.25 * deltaTime;

 for (int i = 0 ; i < material_length ; i++){
   int index = material_map[i];
   real8 szz = - sxx[index] - syy[index] ;

   deltz[index] += quarterDelta * (vnew[index] + v[index]) *
     (    dxx[index] * (sxx[index]  + newSxx[i])  +    dyy[index] * (syy[index]  + newSyy[i]) +
          dzz[index] * (szz            + newSzz[i]) +
      2.*dxy[index] * (txy[index]   + newTxy[i]) + 2.*dxz[index] * (txz[index]  + newTxz[i]) +
      2.*dyz[index] * (tyz[index]   + newTyz[i]) ) ;

   delts[i] += quarterDelta * (vnew[index] + v[index]) *
     (    dxx[index] * sxx[index]  +    dyy[index] * syy[index] +      dzz[index] * szz  +
      2.*dxy[index] * txy[index] + 2.*dxz[index] * txz[index]  +  2.*dyz[index] * tyz[index] ) ;
 }
```

Struct-like access

```
for (int i = 0 ; i < material_length ; i++){
    int index = material_map[i];
    real8 szz = - elem[index].sxx – elem[index].syy ;

    elem[index].deltz += quarterDelta * (elem[index].vnew + elem[index].v) *
      (    elem[index].dxx * (elem[index].sxx + materialElem[i].newSxx)  +
           elem[index].dyy * (elem[index].syy + materialElem[i].newSyy)  +
           elem[index].dzz * (                  szz + materialElem[i].newSzz) +
       2.*elem[index].dxy * (elem[index].txy + materialElem[i].newTxy) +
       2.*elem[index].dxz * (elem[index].txz + materialElem[i].newTxz) +
       2.*elem[index].dyz * (elem[index].tyz + materialElem[i].newTyz) ) ;

    materialElem[i].delts += quarterDelta * (elem[index].vnew + elem[index].v) *
      (    elem[index].dxx * elem[index].sxx  +     elem[index].dyy * elem[index].syy +
           elem[index].dzz * szz                  + 2.*elem[index].dxy * elem[index].txy +
       2.*elem[index].dxz * elem[index].txz  + 2.*elem[index].dyz * elem[index].tyz ) ;
  }
```

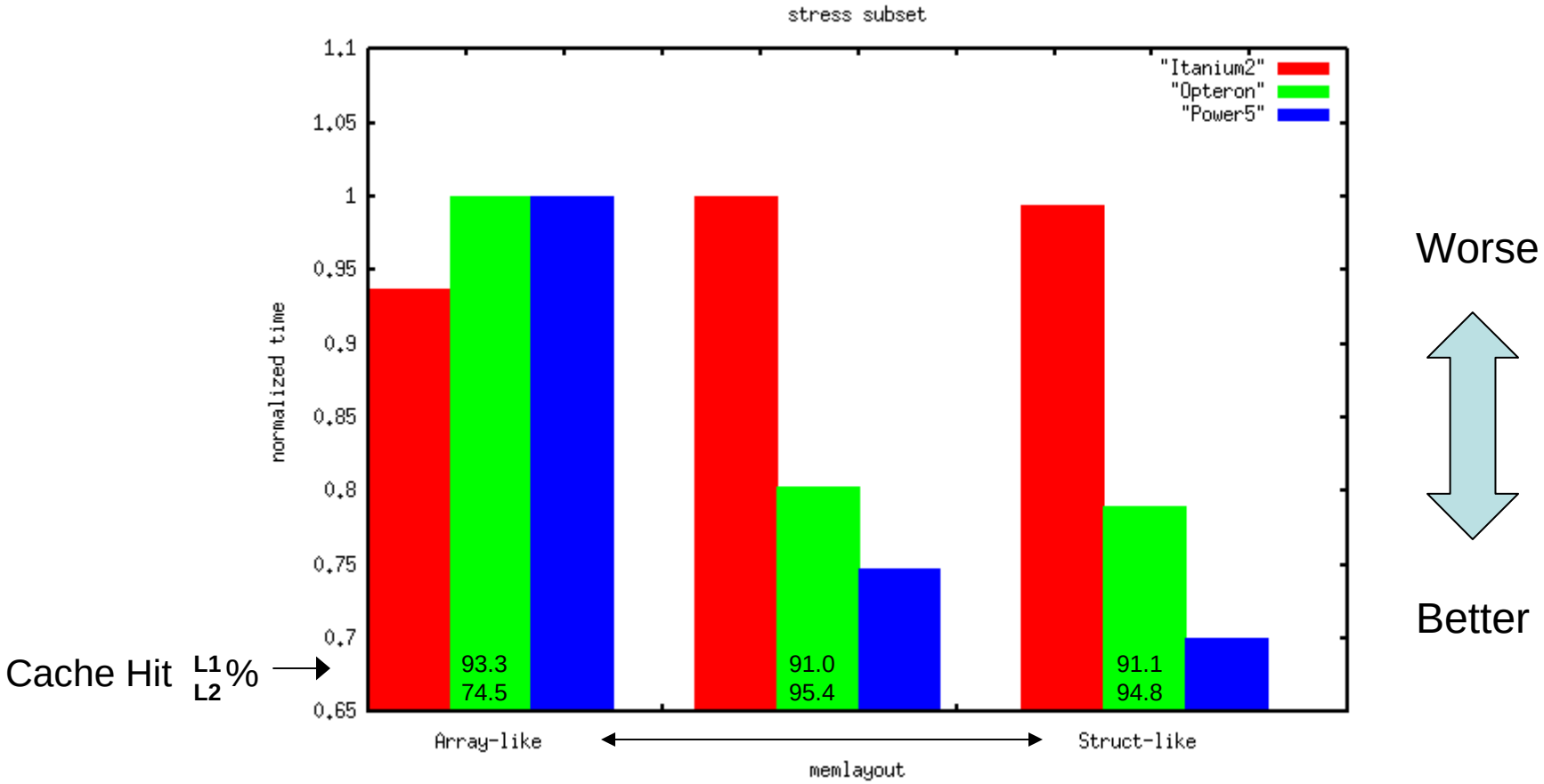
Clustered-Struct access

```
for (int i = 0 ; i < material_length ; i++){
   int index = material_map[i];
   real8 szz = - elem[index].sxx – elem[index].syy ;

   deltz[index] += quarterDelta * (volume[index].vnew + volume[index].v) *
     (    deform[index].dxx * (stress[index].sxx + materialStress[i].newSxx)  +
          deform[index].dyy * (stress[index].syy + materialStress[i].newSyy)  +
          deform[index].dzz * (                    szz + materialStress[i].newSzz) +
      2.*deform[index].dxy * (stress[index].txy + materialStress[i].newTxy) +
      2.*deform[index].dxz * (stress[index].txz + materialStress[i].newTxz) +
      2.*deform[index].dyz * (stress[index].tyz + materialStress[i].newTyz) ) ;

   delts[i] += quarterDelta * (volume[index].vnew + volume[index].v) *
     (    deform[index].dxx * stress[index].sxx  +     deform[index].dyy * stress[index].syy +
          deform[index].dzz * szz                    + 2.*deform[index].dxy * stress[index].txy +
      2.*deform[index].dxz * stress[index].txz  + 2.*deform[index].dyz * stress[index].tyz ) ;
 }
```

Audience Survey

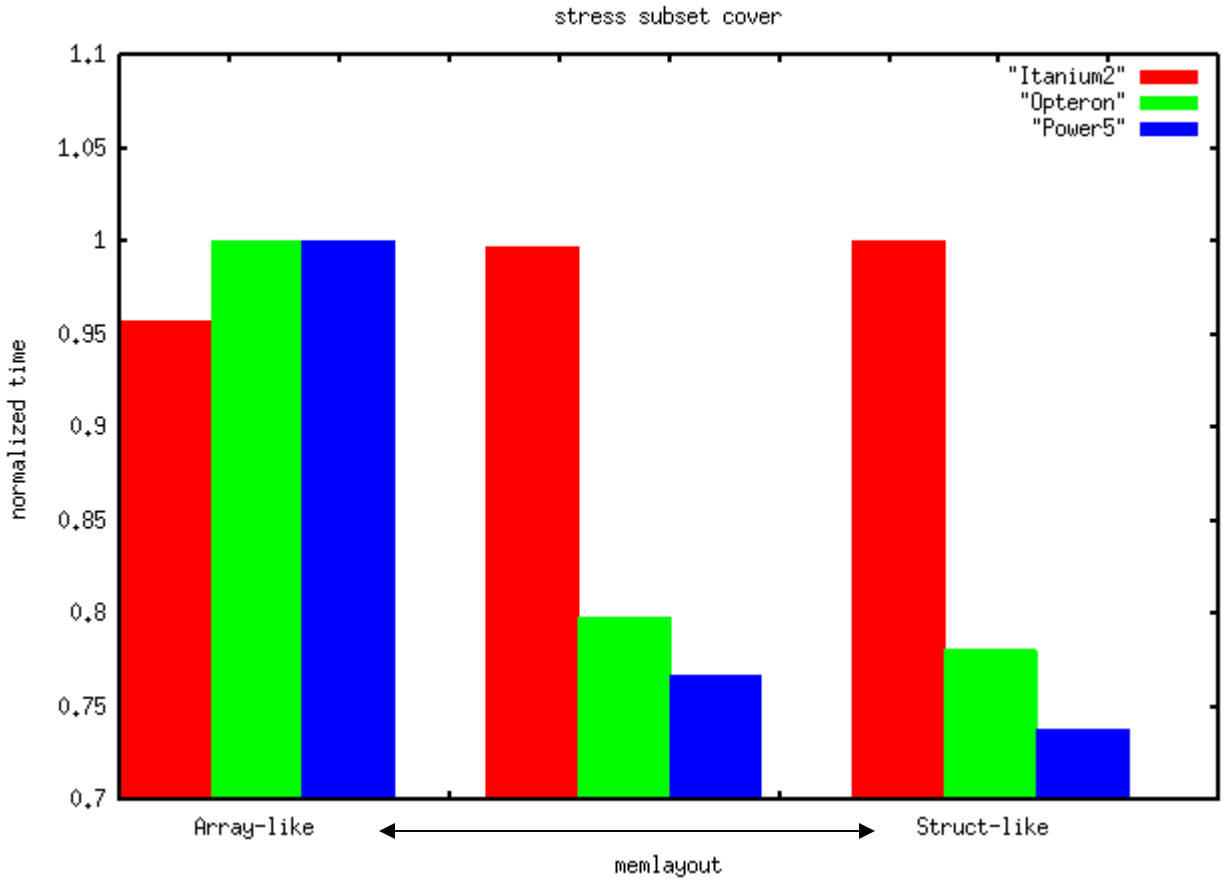
- For those of you who have worked on mesh based software, who has used:
  - An Array-like model?
  - A Struct-like model?
  - A Clustered-Struct Model?
  - Other?
- What model are you using in your software right now?

Performance – Single Region



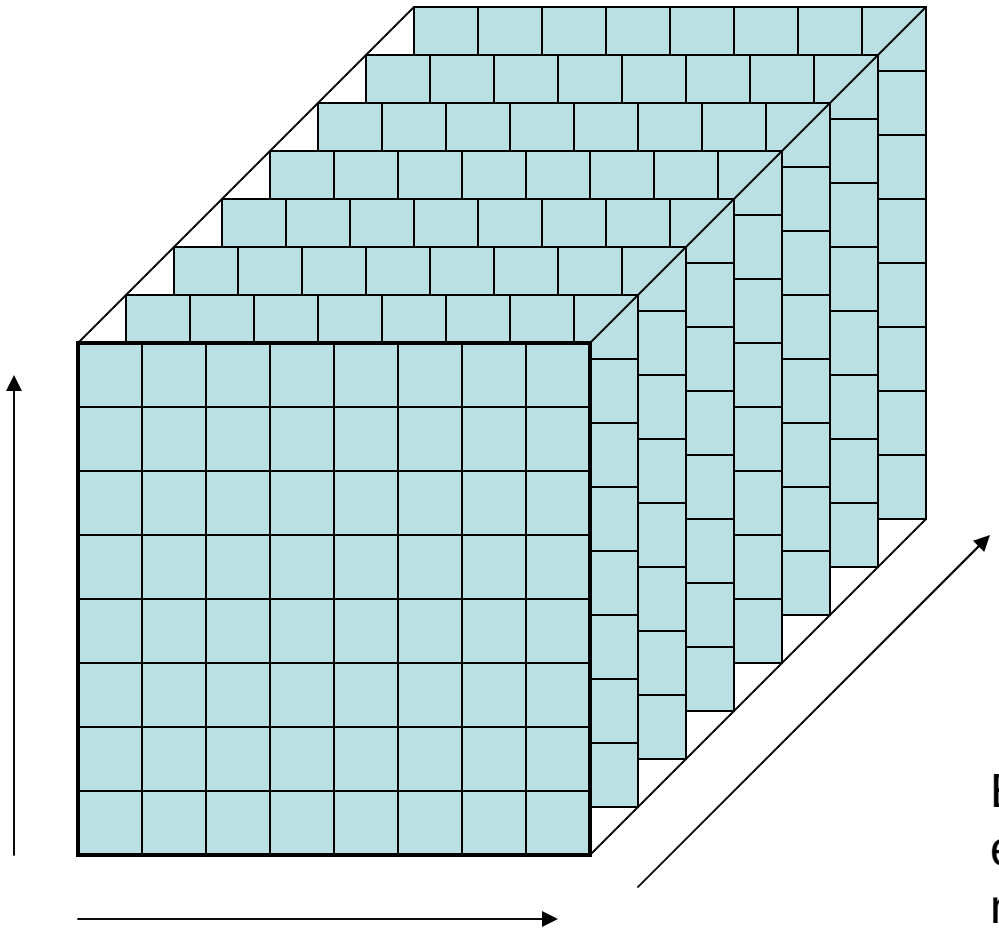
A domain of 12000 elements contains two sparse material subsets of 8000 and 4000 elements. The **8000 element** subset is traversed.

Performance – Mesh Cover



A domain of 12000 elements contains two sparse material subsets of 8000 and 4000 elements. **Both** subsets are traversed, covering the domain.

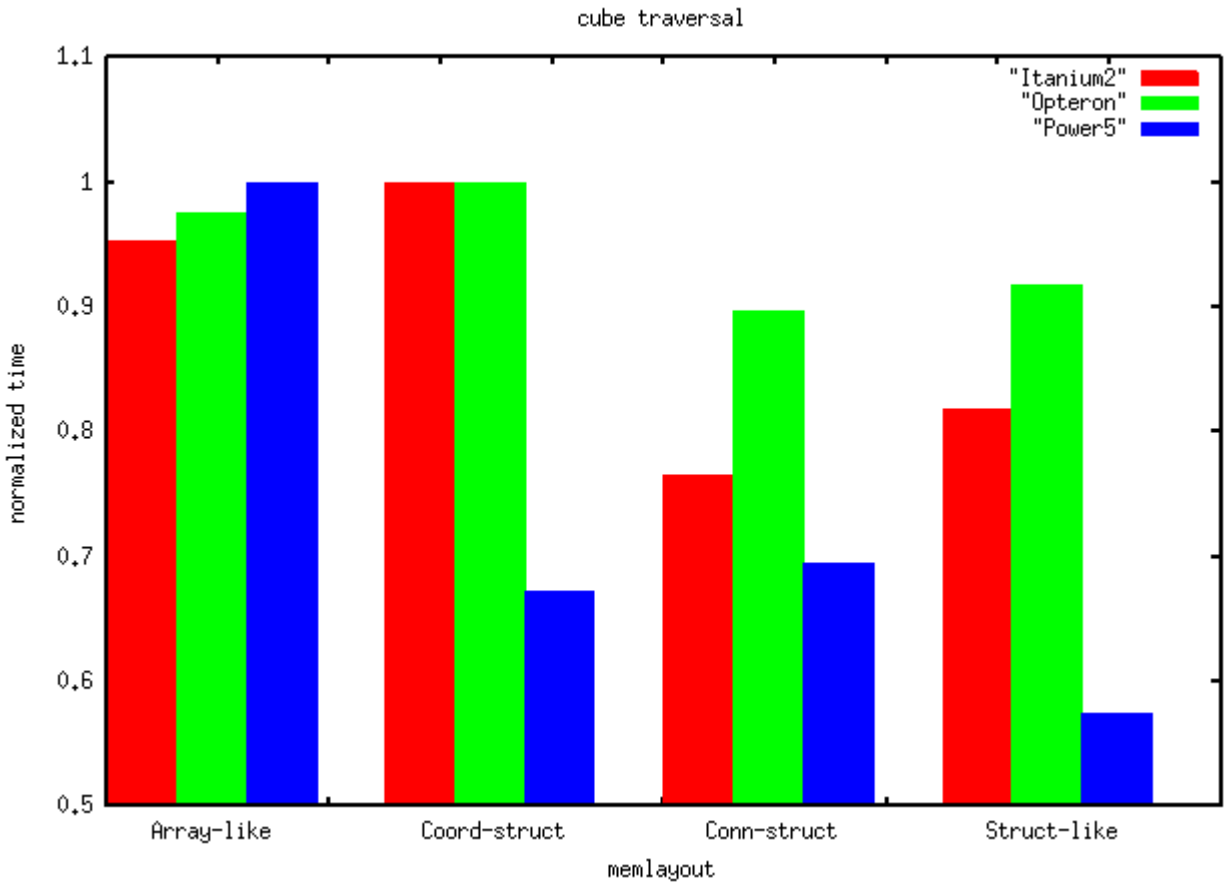
Cube Traversal Example



```
sum = 0.0
foreach (elem) {
  foreach (node) {
    sum += x
    sum += y
    sum += z
  }
}
```

Each element touches eight nodes, and each node has three coordinate components.

Cube Traversal Performance

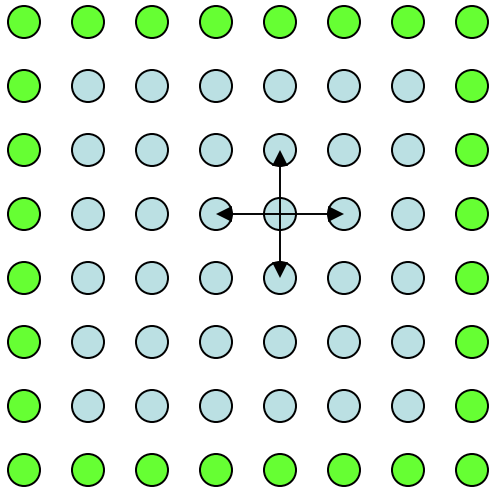


Worse

Better

An unstructured, optimally ordered IndexSet covers a 3D cube of hexahedral elements. Each element accesses eight nodes. Spatial cache reuse.

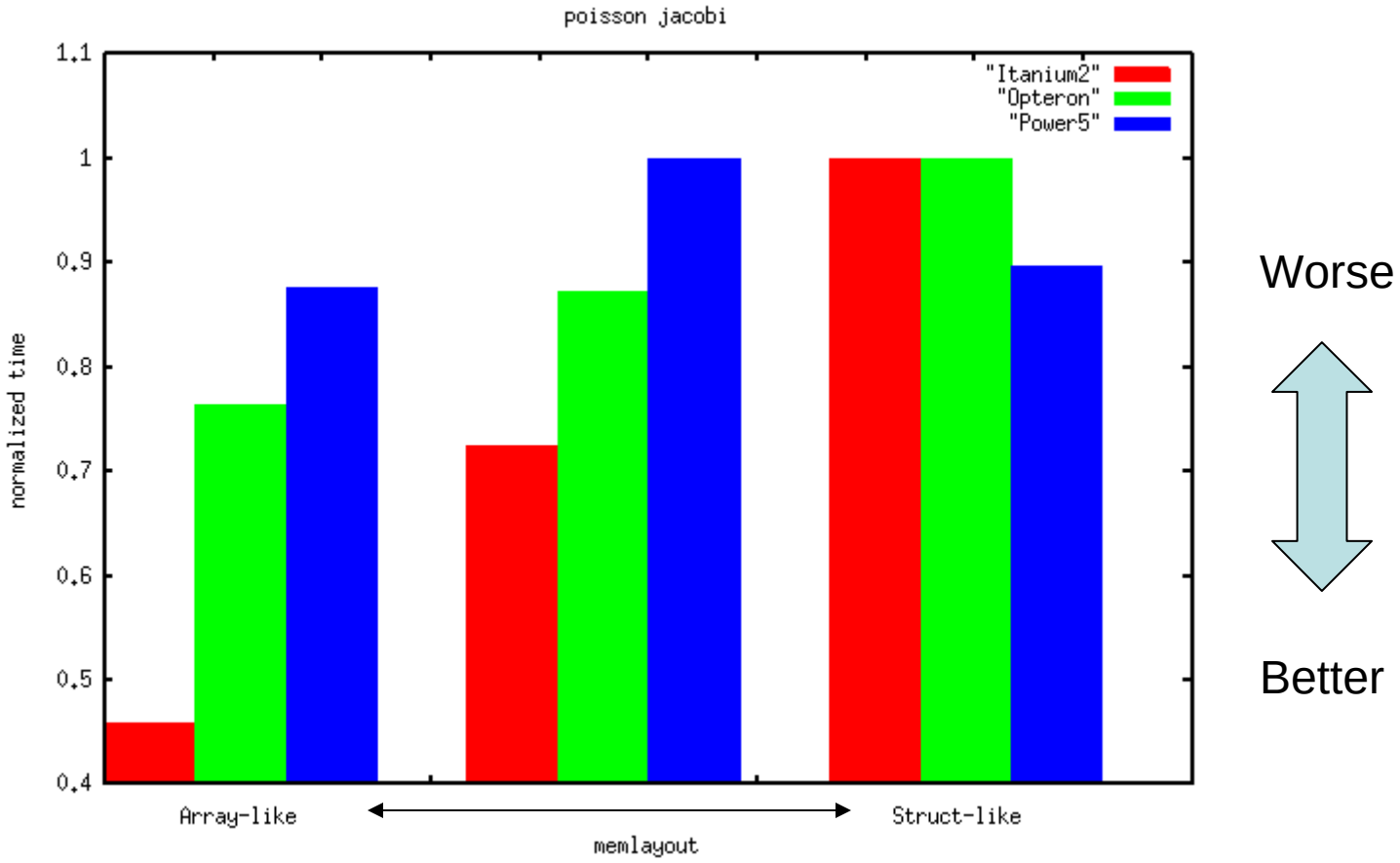
Jacobi Iterative Solution for Poisson Problem



$$\nabla^2 \varphi = f$$

- Boundary, φ = 0
- Interior solution space

Jacobi Iteration Performance



$u\_new = - 0.25 * (f / dh^2 – u\_old(left) –u\_old(right) – u\_old(bottom) – u\_old(top))$

Interpretation of Results

1. Data structure choices are system architecture and problem domain dependent.
2. Data structure choices can result in a 2x performance difference on a given machine.
3. Choosing a data structure that is best for one machine can be the worst for another machine.

Software Engineering Considerations

- There is no simple way to determine the best data structure across all possible architectures at the beginning of a software project.
- Once you have chosen a data structure strategy, it will likely be time-consuming and error-prone to convert to another.
- System architectures change over time.

Hypothetical Case Study

- Assume a large program has been written to use struct-like data structures for memory efficiency.
- Assume a new processor with vector instructions is introduced that performs best with array-like data structures.
- Possible choices:
  1. Rewrite for the array-like data structure so you can take advantage of extra performance on the new architecture with some performance degradation on all other machines.
  2. Keep the struct-like data structure with a significant performance degradation on the new architecture.
  3. Maintain two copies of the code or use preprocessor directives to select between data structures.
  4. Use the best possible data structure on every machine without touching a single line of code…

Vector Extension

```
double quarterDelta = 0.25 * deltaTime;

 while(material) {
   real8 szz = - sxx - syy ;

   deltz += quarterDelta * (vnew + v) *
     (    dxx * (sxx + newSxx) +     dyy * (syy  + newSyy)  +
          dzz * (szz + newSzz) +
      2.*dxy * (txy + newTxy)  + 2.*dxz * (txz + newTxz) +
      2.*dyz * (tyz + newTyz) ) ;

   delts += quarterDelta * (vnew + v) *
     (    dxx * sxx +      dyy * syy +       dzz * szz  +
      2.*dxy * txy  + 2.*dxz * txz  +  2.*dyz * tyz ) ;
 }
```

Vector Schema

- View element
  Field deltz
  Field dxx dyy dzz dxy dxz dyz
  Field sxx syy
  Field txy txz tyz
  Field v vnew
  View material
  Field delts
  Field newSxx newSyy newSzz
  newTxy newTxz newTyz
  View
  View

- Note: Multiple keywords after a **Field** keyword indicate a struct-like interleave.

Vector Extension provides…

- Performance portability across a diversity of system architectures.
- Reduction or elimination of errors due to use of inappropriate indices.
- Simplified tuning of data structures when new algorithms or physics packages are introduced.
- Simplified refactoring.
- **Much** stronger type checking by the compiler than is currently available.
- Enhanced readability of equations.
- Centralization of traversal policy that can be exploited for cache blocking and data movement on a variety of architectural models (multi-core, NUMA, GPGPU).

Vector Extension Drawbacks

- Native debugging requires vector extension support by mainstream compilers (or direct use of intermediate source).
- Level of abstraction is increased (although not by much).

Transition to Vector Extension

- Using the vector extension requires four steps:
  - Creating a schema to identify topological relationships among Fields in your software (doubles as good documentation).
  - Stripping indices off of arrays in loops (can almost be done by a sed script).
  - Changing **for** loops over index variables into **while** loops over IndexSets.
  - Centralizing memory allocation for discrete Field variables of interest. Many projects at the lab already meet this requirement (wrapped malloc, database, etc.).

Semi-Automatic “Vectorization”

- Existing software that conforms to certain restrictions might be transformable to vector extended code with limited user intervention.
- ROSE could help check for conformance issues:
  - Tell user where non-resolvable conformance issues exist.
  - Correct simple conformance mistakes.
- This would allow users who are uncomfortable with vector abstractions to implement and debug their software in their current language, while still gaining the advantages of using the vector extended compiler.

Conclusion

- A diversity of hardware architectures are being introduced simultaneously (Multi-core, NUMA, GPGPU/vector coprocessors).
- A low-impact change in our programming model may provide a unified way of running effectively on a diversity of system architectures.
- A vector compiler has been written using ROSE and is available for anyone wanting to experiment with native vector extended code.
- The potential value of this approach is reflected in the presented results. It would be nice to go further.

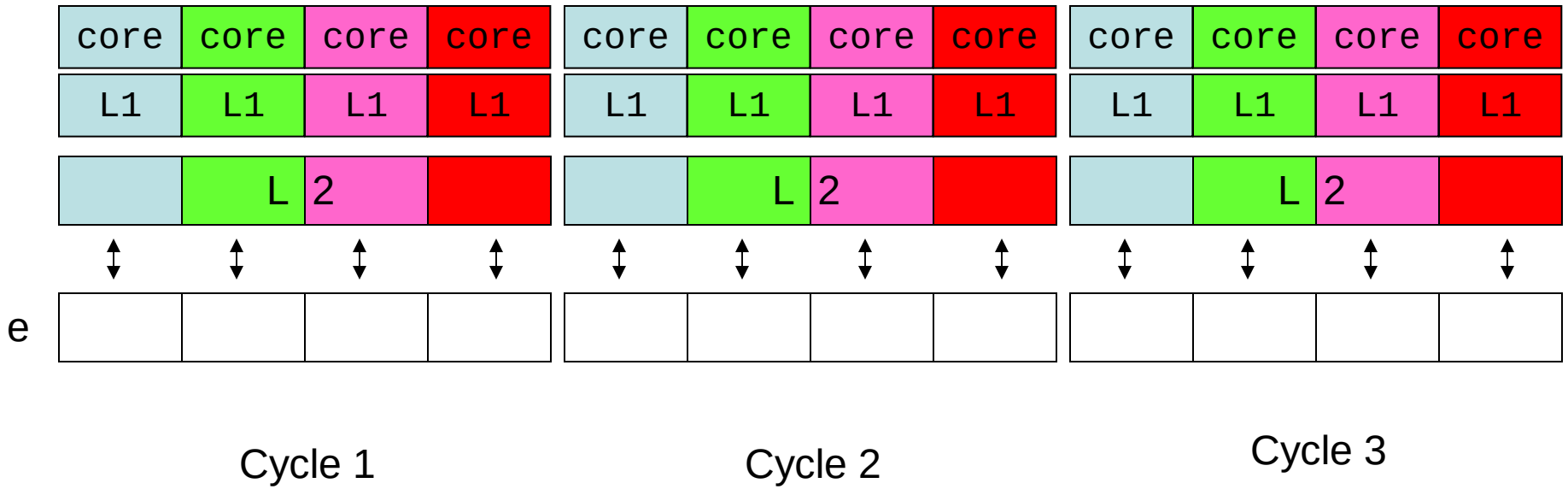
Cache Performance

	Opteron Hardware Counters L1 Cache		
Memory Interleave	Hit Count	Miss Count	Hit Ratio
Array-like	3955732080	286239697	93.3%
Intermediate	2842569424	281404535	91.0%
Struct--like	2769568352	273753504	91.1%

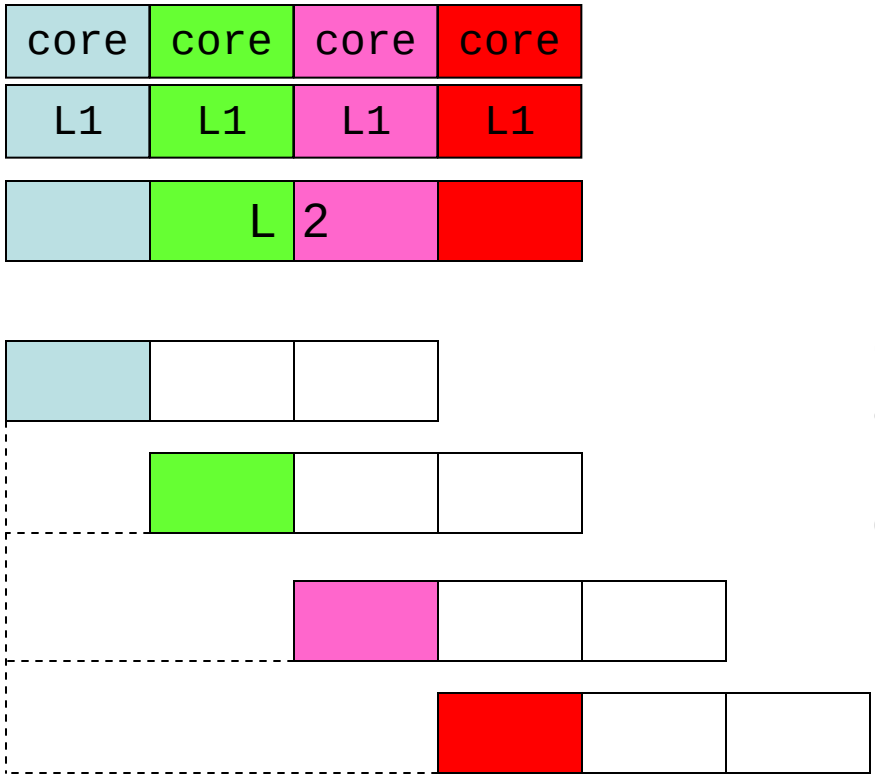
Some Correctness Features

- Removal of explicit subscripting can reduce or eliminate indexing errors.
- Nonsensical Field operations are easily caught by the compiler in equations because every Field used in an equation must either:
  - be a topological sibling, ancestor, or descendant.
  - be in a subspace that can be accessed through an appropriate Relation array.
- Improved bounds checking.
- Consistent Field naming throughout the code (because of schema template) allows for enhanced readability and argument checking at caller/callee boundaries.

Multi-core Cache -- Threads

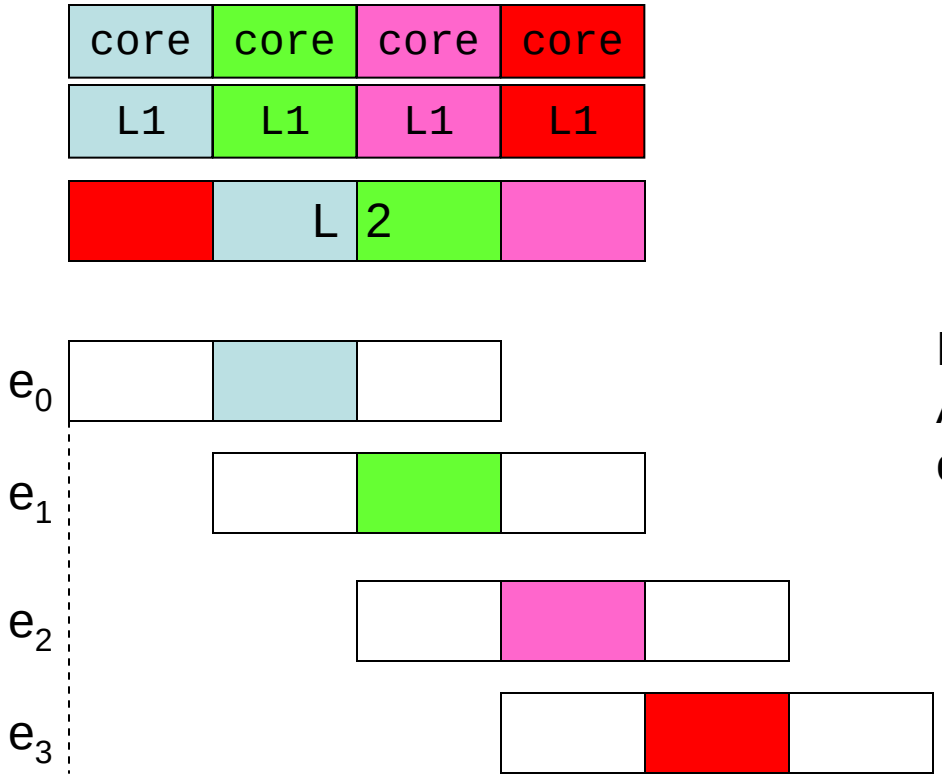


Multi-core Cache – MPI Cycle 1



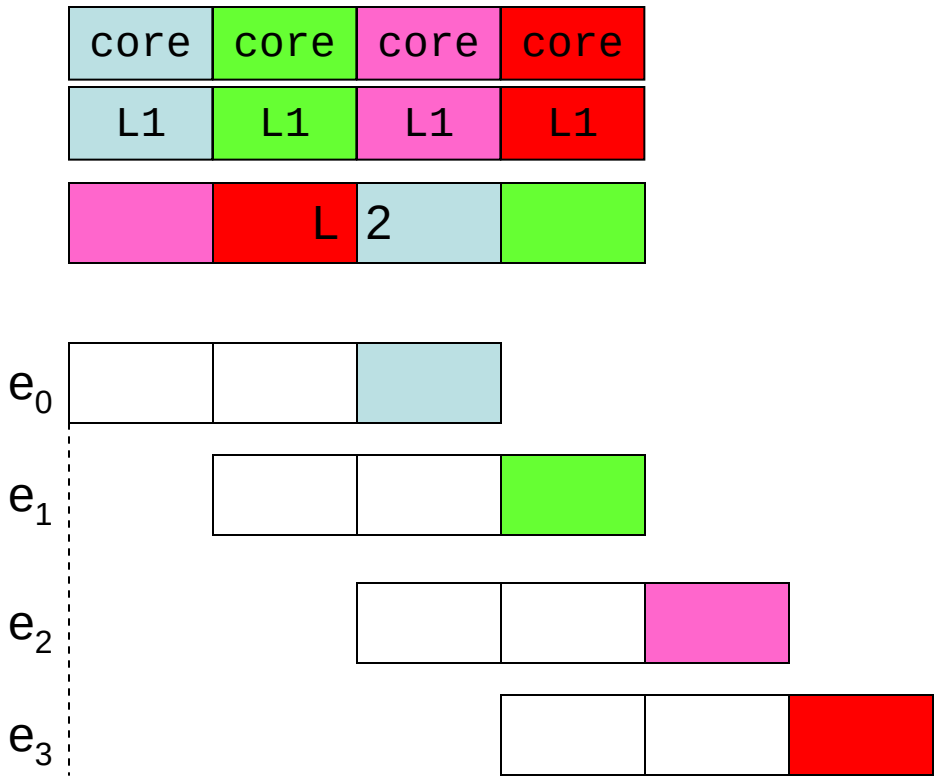
Custom pool memory allocation allows a vector in separate MPI processes running on the same chip to be cache aligned to a particular core.

Multi-core Cache – MPI
Cycle 2

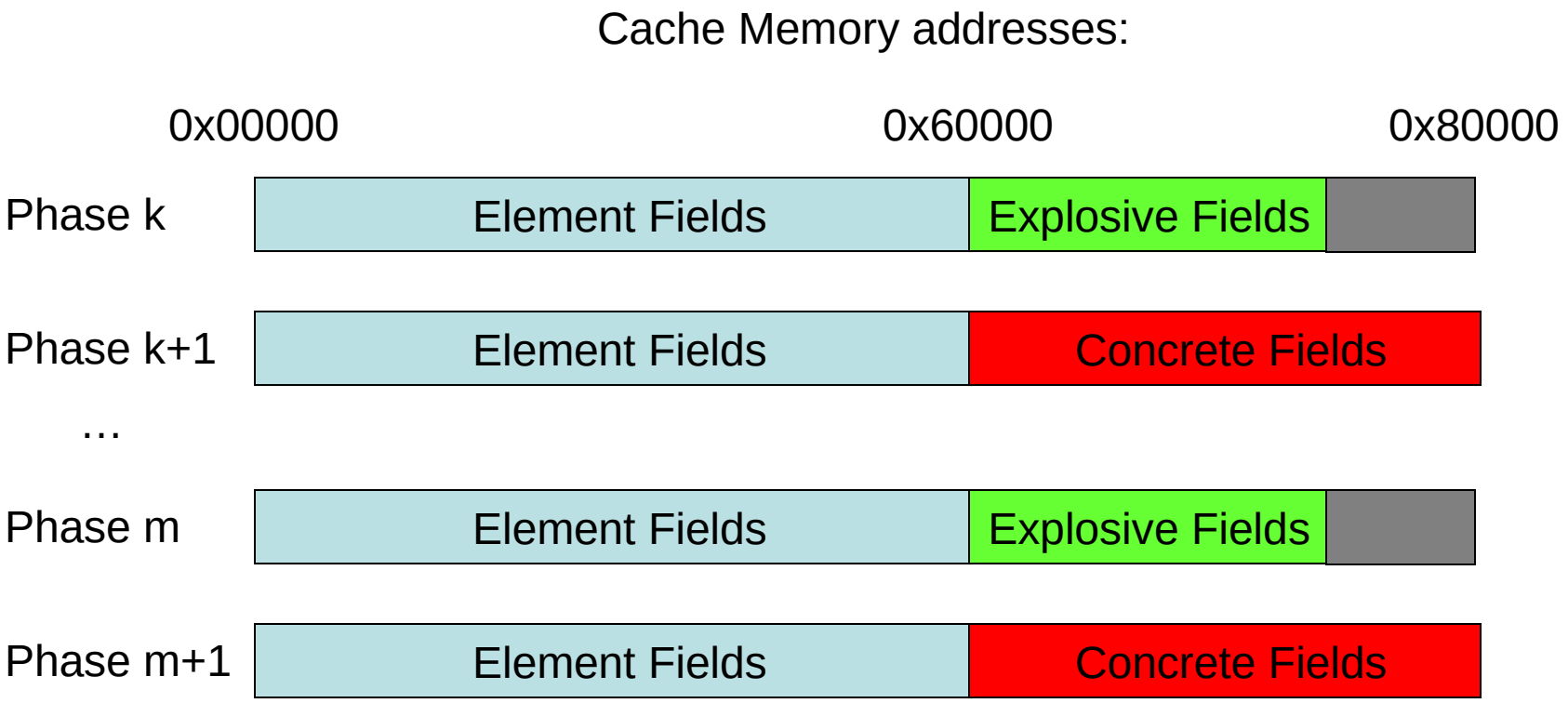


Each core works on its own array.
An L2 cache slice will be cyclically owned by each core.

Multi-core Cache – MPI
Cycle 3



Finding an “Optimal” Schema



Search space for optimal schema is narrowed. Rather than having a combinatorial problem over the space of all Fields, you have a combinatorial problem over many small independent subspaces.